SEMANA 3

Academia Xideral

3 DE FEBRERO DE 2023

DANIEL DE JESUS MOLINA GARCIA

Índice

Contenido

1 Spring Batch

Es un marco de procesamiento por lotes que es utilizado para trabajar con volúmenes muy grandes de datos y generalmente de una forma programada. Es decir, sin intervención humana, haciendo la aclaración que no se trata de un planificador de tareas, aunque estos también pueden llegar a integrarse con Spring Batch.

Un Spring Batch suele tener los siguientes componentes:

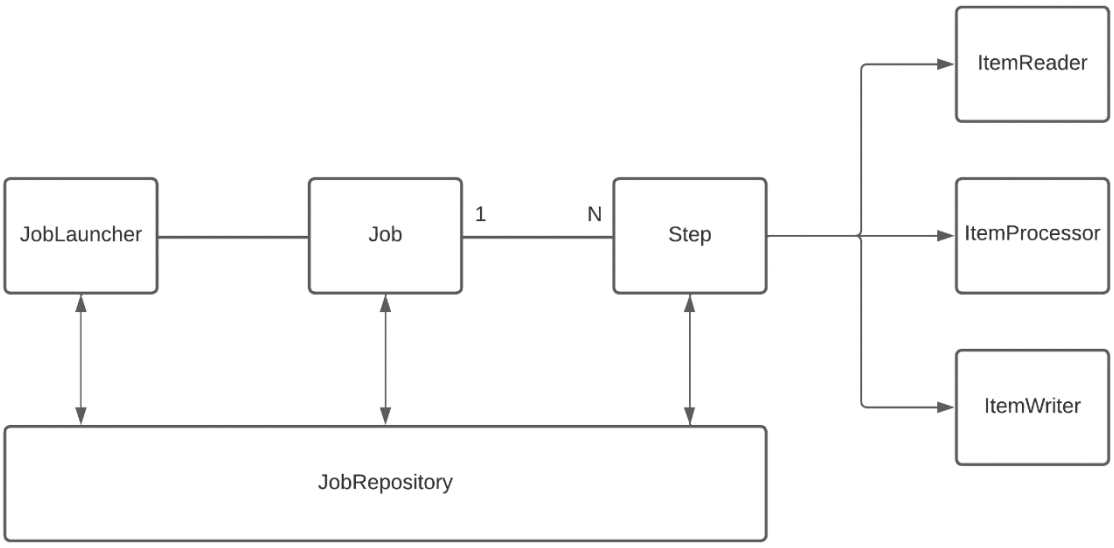
JOB-REPOSITORY: Es el mecanismo responsable de almacenar información sobre cada Job, Step que se produzca, los parámetros del Job, así como también qué procesos son los que se están ejecutando, cuantos y los estados de cada uno, los errores que tengan lugar, etc.

JOB-LAUNCHER: Los JobLaunchers son responsables de iniciar un Job dentro del sistema con determinados parámetros.

JOB y STEP: Un Job es un proceso y está compuesto por uno o varios pasos o Steps. Una vez se han llevado a cabo todos estos pasos, se considera el Job como completado.

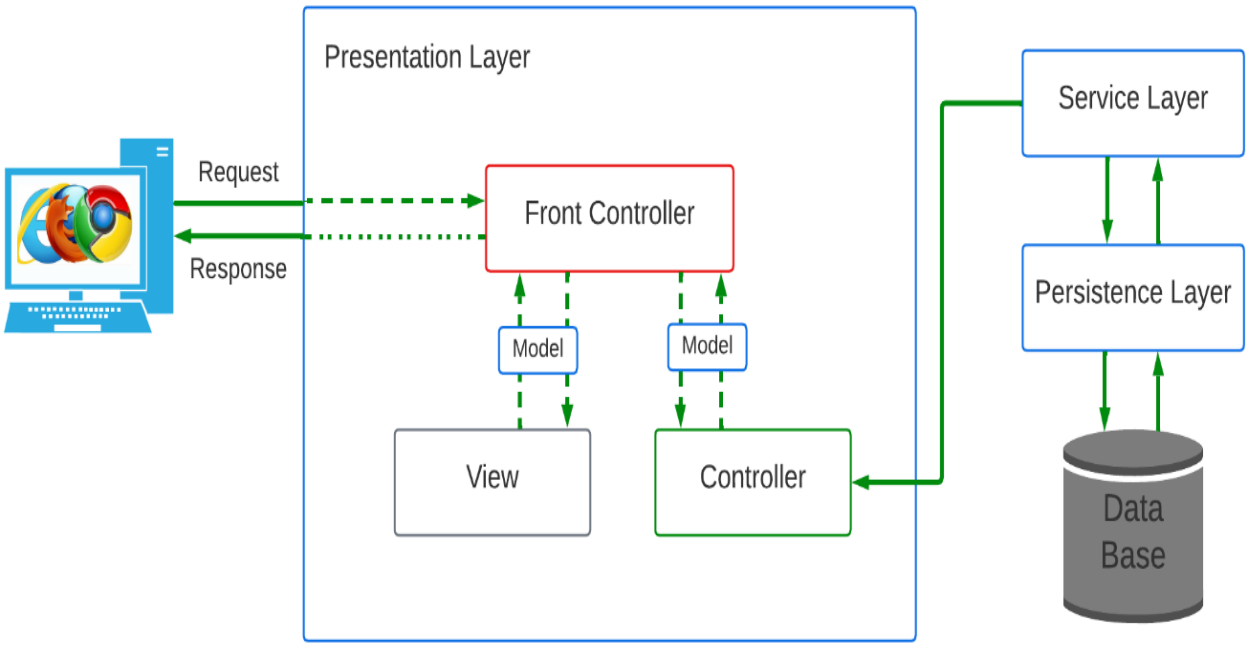
Cada uno de estos Steps suelen estar compuestos de tres elementos:

- **ItemReader**: En esta fase, los datos se leen desde un origen, como un archivo o una base de datos.
- **ItemProcessor**: Los datos se procesan de alguna manera como validación, la transformación o la agregación.
- **ItemWriter**: Los datos se escriben en un destino, como un archivo o una base de datos.



Haciendo un pequeño resumen de cómo funciona un Spring Batch, como se muestra en nuestro ejemplo, un proceso por lotes generalmente está encapsulado por un Job que consta de múltiples Steps. Cada Step normalmente tiene uno o varios ItemReader, ItemProcessor y ItemWriter. Un Job es ejecutado por un JobLauncher, y los metadatos sobre los trabajos configurados y ejecutados se almacenan en un JobRepository.

2 Diagrama de Spring MVC



3 GITHUB: Comandos Avanzados

3.1 Git pull request

Se utiliza para colaborar de manera efectiva con otros desarrolladores y revisar el código antes de que se integre en la rama principal del proyecto y garantizar la calidad del código antes de que se integre en el proyecto principal.

Para crear un pull request, primero debes crear una rama en tu repositorio local y hacer tus cambios en esa rama. Luego, puedes enviar una solicitud para que tus cambios sean revisados y fusionados con la rama principal del proyecto.

Los mantenedores del proyecto pueden revisar el código, hacer comentarios y discutir cualquier cambio antes de aceptar o rechazar la solicitud de pull. Si la solicitud es aceptada, los cambios se fusionan con la rama principal y se incluyen en el proyecto.

3.2 Git fork

se refiere a la creación de una copia de un repositorio en GitHub o en otra plataforma de control de versiones. Este proceso permite a un usuario tener su propia versión del repositorio y hacer cambios sin afectar el repositorio original.

Para "hacer un fork" de un repositorio en GitHub, un usuario puede ir a la página del repositorio y hacer clic en el botón "Fork". Esto crea una copia exacta del repositorio original en el perfil del usuario. Desde allí, el usuario puede hacer cambios en su copia del repositorio y enviar solicitudes de pull para integrar esos cambios en el repositorio original.

3.3 Git rebase

se utiliza para reorganizar las operaciones de confirmación en una rama. A diferencia de git merge, que crea una nueva confirmación con la fusión de dos ramas, git rebase se utiliza para reaplicar todas las confirmaciones de una rama sobre otra rama.

El objetivo principal de git rebase es hacer que la historia de confirmaciones sea más lineal y fácil de entender, eliminando las ramificaciones en la historia. Por

ejemplo, si tienes una rama con varias confirmaciones y la haces diverger de la rama principal, puedes usar git rebase para aplicar esas confirmaciones sobre la rama principal actualizada en lugar de crear una nueva confirmación de fusión.

Sin embargo, el uso de git rebase también puede ser peligroso, ya que puede causar conflictos si hay cambios en la rama principal que entran en conflicto con tus confirmaciones. Por lo tanto, es importante utilizar git rebase con cuidado y asegurarse de entender sus implicaciones antes de utilizarlo.

3.4 Git stash

se utiliza para guardar temporalmente cambios no confirmados en un repositorio. Esto es útil cuando tienes cambios en curso y necesitas cambiar a otra rama o hacer una operación que requiere una copia limpia del repositorio. En lugar de confirmar o descartar los cambios, puedes guardarlos temporalmente con git stash.

El comando git stash guarda los cambios en un "paquete de desecho", que puedes aplicar más tarde en cualquier momento. Por ejemplo, si tienes cambios en curso y quieres cambiar a otra rama para resolver un problema urgente, puedes usar git stash para guardar tus cambios en un paquete de desecho, cambiar a la otra rama y resolver el problema. Después de haber resuelto el problema, puedes regresar a la rama original y aplicar los cambios guardados en el paquete de desecho.

El comando git stash también permite opciones adicionales para aplicar o descartar los cambios guardados en el paquete de desecho. Por ejemplo, puedes usar git stash apply para aplicar los cambios guardados o git stash drop para descartarlos permanentemente.

3.5 Git clean

se utiliza para eliminar archivos no seguidos de un directorio de trabajo. Esto significa que git clean elimina cualquier archivo o directorio que no esté incluido en el control de versiones y que no esté en seguimiento.

El comando git clean es útil cuando quieres limpiar tu directorio de trabajo y eliminar archivos que no son relevantes para el control de versiones. Por ejemplo, puedes

usar git clean después de compilar tu proyecto para eliminar los archivos generados durante la compilación.

Es importante tener en cuenta que git clean elimina archivos permanentemente y no pueden ser recuperados. Por lo tanto, es importante usar git clean con cuidado y asegurarse de que los archivos que se van a eliminar son realmente innecesarios antes de ejecutar el comando.

3.6 Git cherry pick

se utiliza para aplicar uno o más commits específicos a la rama actual. Con git cherry-pick, puedes seleccionar uno o más commits de otra rama y aplicarlos a la rama actual, sin tener que fusionar la otra rama en su totalidad.

es útil cuando quieres aplicar cambios de una rama a otra sin tener que fusionar las ramas completamente. Por ejemplo, si tienes una rama de características que incluye un conjunto de cambios y quieres aplicar solo unos pocos de esos cambios a la rama principal, puedes usar git cherry-pick para seleccionar esos cambios específicos y aplicarlos a la rama principal.

Es importante tener en cuenta que git cherry-pick puede causar conflictos si los cambios aplicados entran en conflicto con los cambios existentes en la rama actual. Por lo tanto, es importante revisar cuidadosamente los resultados de git cherry-pick antes de confirmar los cambios.